Building Assembler Programs

Chapter 5
Sections 1-6

Dr. Iyad Jafar
Outline

- Building Structured Programs
- Conditional Branching
- Subroutines
- Generating Time Delays
- Dealing with Data
- Example Programs
Writing programs is not an easy task; especially with large and complex programs.

It is essential to put down a design for the program before writing the first line of code.

This involves documenting the programs flow charts and state diagrams.
Building Structured Programs

- **Flowcharts**
  - Rectangle for process
  - Diamond for decision
Building Structured Programs

- **State Diagrams**
  - Circle for state
  - Arrow for state transition labeled with condition(s) that causes the transition

![State Diagram](image.png)
Conditional Branching

- Microprocessors and microcontroller should be able to make decisions
- This enables them to behave according to the state of logical variables
- The PIC 16 series is not an exception! They have *four conditional skip* instructions
- These instructions *test for a certain condition and skip the following instruction* if the tested condition is true!
## Conditional Branching

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>btfsc f, b</td>
<td>Test bit at position b in register f. Skip next instruction if the bit is clear ‘0’</td>
<td>btfsc STATUS, 5</td>
</tr>
<tr>
<td>btfss f, b</td>
<td>Test bit at position b in register f. Skip next instruction if the bit is set ‘1’</td>
<td>btfss 0x21, 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>decfsz f, d</td>
<td>Decrement the contents of register f by 1 and place the result in W if d = 0 or in f if d = 1. Skip next instruction if the decremented result is zero</td>
<td>decfsz 0x44, 0</td>
</tr>
<tr>
<td>incfsz f, d</td>
<td>Increment the contents of register f by 1 and place the result in W if d = 0 or in f if d = 1. Skip next instruction if the incremented result is zero</td>
<td>incfsz 0xd1, 1</td>
</tr>
</tbody>
</table>
Conditional Branching

- **Example 1**: a program to add two numbers in locations 0x11 and 0x22. If there is no carry, store the result in location 0x33, else store the result in location 0x44.

- Reminder: the STATUS Register

<table>
<thead>
<tr>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R-1</th>
<th>R-1</th>
<th>R/W-x</th>
<th>R/W-x</th>
<th>R/W-x</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRP</td>
<td>RP1</td>
<td>RP0</td>
<td>TO</td>
<td>PD</td>
<td>Z</td>
<td>DC</td>
<td>C</td>
</tr>
</tbody>
</table>

(bit 7) (bit 0)
Conditional Branching
Example 1

STATUS equ 0x03 ; define SFRs
org 0x0000 ; reset vector
goto START
org 0x0006

START
movf 0x11 , 0 ; get first number to W
addwf 0x22 , 0 ; add second number
btfsc STATUS , 0 ; check if carry is clear
goto C_SET ; go label C_Set if C==1
movwf 0x33 ; store result in 0x33

C_SET
movwf 0x44

DONE
end

goto DONE ; endless loop
Conditional Branching

- **Example2**: assume a 16-bit counter with upper byte in location COUNTH and lower byte in location COUNTL. Write the code to decrement the counter until it is zero.
Conditional Branching Example 2

COUNTL equ 0x10 ; lower byte of counter in 0x10
COUNTH equ 0x11 ; upper byte of counter in 0x11
#include "P16F84A.INC"

org 0x0000

START
    movf COUNTL, F ; check if the both locations are zeros
    btfss STATUS, Z ; if so, then finish
    goto DEC_COUNTL ; if COUNTL is not zero, decrement it
    movf COUNTH, F ; if it is zero check COUNTH
    btfsc STATUS, Z
    goto DONE ; if both are zeros, then DONE
    decf COUNTH, F

DEC_COUNTL
    decf COUNTL, F
    goto START

DONE
    goto DONE ; program gets here if both are zeros
end
Subroutines

- In many cases, we need to use a block of code in a program in different places.
- Instead of writing the code wherever it is needed, we can use subroutines/functions/procedures.
  - Blocks of code saved in memory and can be called/used from anywhere in the program.
  - When a subroutine is called, execution moves to place where the subroutine is stored.
  - Once the subroutine is executed, execution resumes from where it was before calling the subroutine.
Subroutines

Do this
Do that
Do something else
Call SR1
Do that
......
......
......
Call SR2
......
......
Call SR1
......
......
......
......
SR1
......
......
......
......
Return

Subroutine 2

......
......
......
......
......
......
Return

SR2
Subroutines

- The *program counter* holds the address of the instruction to be executed.
- In order to call a subroutine, the program counter has to be loaded with the address of the subroutine.
- Before that, the current value of the PC is saved in *stack* to assure that the main program can continue execution from the following instruction.
Subroutines

- In PIC, to invoke a subroutine we use the **CALL** instruction followed by the address of the subroutine.

- The address is usually specified by a symbolic label in the program.

- To exit a subroutine and return to the main program, we use the **RETURN** or **RETLW** instructions.
Subroutines - Example

; A subroutine to perform multiplication between locations 0x30 and 0x31. the result is
; returned in the working register.
STATUS equ 0x03 ; define SFRs
org 0x0000 ; reset vector
goto START
org 0x0005

START
......
movlw 0x15 ; pass the first number
movwf 0x30
movlw 0x09 ; pass the second number
movwf 0x31
call multiply ; call the subroutine
......
movlw 0x05 ; pass the first number
movwf 0x30
movlw 0x04 ; pass the second number
movwf 0x31
call multiply ; call the subroutine
......
DONE goto DONE ; endless loop
Example - Continued

```
multiply
Repeat
  clrw
  addwf 0x30, 0 ; repeated addition
  decfsz 0x31, 1 ; counter
  goto repeat
return
end
```
Generating Time Delays

- In many applications, it is required to delay the execution of some block of code; i.e., a time delay!
- In most microcontrollers, this can be done by
  - Software
  - Hardware (Timers)
- To generate time delay using software, let the microcontroller execute non-useful instructions for certain number of times!
- If we know the clock frequency and the cycles to execute each instruction, we can generate different delays

\[
\text{Delay} = \#\text{cycles} \times \text{clock cycle time}
\]

\[= \#\text{cycles} \times \frac{4}{F_{osc}}\]
Generating Time Delays

- Structure of Delay Loops

One loop for small delays

Nested loops for large delays
Generating Time Delays

- Example 1: Determine the time required to execute the following code. Assume the clock frequency is 800KHz.

```
movlw D’200’ ; initialize counter
movwf COUNTER

; main loop for delay
nop
nop
decfsz COUNTER, F
goto del
```

- What if this code to be used as a subroutine??!!
Generating Time Delays

- Example 2: analyze the following subroutine and show how it can be used to generate a delay of 10 ms exactly including the call instruction. Assume 4 MHz clock frequency

```
TenMs  nop ; beginning of subroutine
movlw D'13'
movwf COUNTH
movlw D'250'
movwf COUNTL
Ten1  decfsz COUNTL, F ; inner loop
goto Ten1
decfsz COUNTH, F ; outer loop
goto Ten1
return
```
call TenMs
nop
movlw 1   ; (TenMsH)
movwf COUNTL
movlw 250 ; (TenMsL)
movlw COUNTL

decfsz COUNTL,F  
  decfsz COUNTL,F  
  decfsz COUNTH,F  
  goto Ten_1

COUNTL: 250→249→...→2→1
3 x 249 = 747

COUNTL: 0→255→254→...→2→1
3 x 255 = 765

COUNTL: 1→0
770** x 11 = 8470

COUNTH: 12→11
2

COUNTH: 10→12
1

770

Repeat this block eleven times as COUNTH: 12→11→...→2→1

COUNTL: 0→255→254→...→2→1
3 x 255 = 765

COUNTL: 1→0
2

COUNTH: 1→0
2

return

Total = 10,000
Working with Data

Indirect Addressing

- Direct addressing is capable of accessing single bytes of data
- Working with list of values using direct addressing is inconvenient since the address is part of the instruction
- Instead, we can use indirect addressing where
  - the File Select Register FSR register acts as a pointer to data location.
  - The FSR can be incremented or decremented to change the address
- The value stored in FSR is used to address the memory whenever the INDF (0x00) register is accessed in an instruction
- This forces the CPU to use the FSR register to address memory
Working with Data

Direct/Indirect Addressing in 16F84A

Note 1: For memory map detail, see Figure 2-2.
2: Maintain as clear for upward compatibility with future products.
3: Not implemented.
Working with Data

Example: a program to add the values found locations 0x10 through 0x1F and store the result in 0x20

```
STATUS equ 0x03 ; define SFRs
FSR equ 0x04
INDF equ 0x00
RESULT equ 0x20
N equ D'15'
COUNTER equ 0x21
org 0x0000 ; reset vector
goto START
org 0x0005

START
  movlw N ; initialize counter
  movwf COUNTER
  movlw 0x11 ; initialize FSR as a pointer
  movwf FSR
  movf 0x10 , W ; get 1st number in W

LOOP
  addwf INDF , W ; add using indirect addressing
  incf FSR , F ; point to next location
decfsz COUNTER , F ; decrement counter
goto LOOP

DONE
  movwf RESULT
  goto DONE

end
```
Working with Data

Look-up Tables

- A look-up table is a block of data held in the program memory that the program accesses and uses.
- The `movlw` instruction allows us to embed one byte within the instruction and use! How about a look-up table?
- In PIC, look-up tables are defined as a subroutine inside which is a group of `retlw` instructions.
- The `retlw` instruction is similar to the return instruction; however, it has one operand which is an 8-bit literal that is placed in `W` after the subroutine returns.
- In order to choose one of the `retlw` instructions in the look-up table, the program counter is modified to point to the desired instruction by changing the value in the `PCL` register (0x02).
- The `PCL` register holds the lower 8 bits of the program counter.
Example: A subroutine to implement a look-up table for the squares for number 0 through 5. To compute the square, place the number in W before calling the subroutine SQR_TABLE.

```asm
SQR_TABLE   addwf PCL , 1 ; modify the PCL to point the
            ; required instruction
retlw D'0' ; square value of 0
retlw D'1' ; square value of 1
retlw D'4' ; square value of 2
retlw D'9' ; square value of 3
retlw D'16' ; square value of 4
retlw D'25' ; square value of 5
```

; Remember that the PC always points to the instruction to be executed
Summary

- Building complex programs requires putting down its requirements and design.

- Programs tend to execute instructions sequentially unless branching or subroutines are used.

- A subroutine is a piece of code that can be called from anywhere inside the program.

- A simple way to generate time delays is to use delay loops.